Chapter 1

Developing Spectacular Android Applications

COPYRIGHTED MATERIAL

In This Chapter

- Seeing reasons to develop Android apps
- Starting with the basics of Android programming
- Working with the hardware
- Getting familiar with the software

Google rocks! Google acquired the Android project in 2005 (see the sidebar "The roots of Android" later in this chapter) to ensure that a mobile operating system (OS) could be created and maintained in an open platform. Google continues to pump time and resources into the Android project, which has already proved to be beneficial. As of July 2010, 160,000 Android handsets have been activated daily, which is good considering that handsets have been available only since October 2008. That's less than two years, and Android has already made a huge impact!

It has never been easier for a developer to be able to make money on his own. Android users may not know who you are, but they know what Google is, and they trust Google. Because your app resides in the Android Market — which Google controls — Google assumes that your application is okay too.

Why Develop for Android?

Well, the real question should be "Why not?" Do you want your app to be available to millions of users worldwide? Do you want to publish apps as soon as you're done writing and testing them? Do you like developing on open platforms? If you answered yes to any of these questions, I think you have your answer, but in case you're still undecided, keep reading, and I'll explain what I mean.

Market share

As a developer, you have an opportunity to develop apps for a fairly new market that is booming on a daily basis. Android is currently set to outpace many other carriers in market share in the industry in coming months. With so many users, it's never been easier to write an application that can be downloaded and used by real people! The Android Market puts your app right into your users' hands easily! Users don't have to go searching the Internet to find an app to install. They just simply go to the Android Market that is preinstalled on their device, and they have access to all *your* apps. Because the Android Market comes preinstalled on most Android devices (I discuss a few exceptions later), users typically search the Android Market for all of their app needs. It's not hard to see an app's number of downloads soar in just a few days.

Time to market

With all the application programming interfaces (APIs) that Android comes packed with, it's easy to develop full-featured applications in a relatively short time frame. After you've signed up with the Android Market, just upload your apps and publish them. "Wait," you may say, "are you sure?" Why, yes, I am! Unlike other mobile marketplaces, the Android Market has no app-approval process. All you have to do is write apps and publish them.

Technically, anyone can publish anything, but it's good karma to keep within Google's terms of service and keep your apps family-friendly. Remember that Android users come from diverse areas of the world and are in all age categories.

Open platform

The Android operating system is *open platform,* meaning that it's not tied to one hardware manufacturer and/or one provider. As you can imagine, the openness of Android is allowing it to gain market share quickly. All hardware manufacturers and providers can make and sell Android devices. The Android source code is available at `http://source.android.com` for you to view and/or modify. Nothing is holding you back from digging into the source code to see how a certain task is handled. The open-source code allows phone manufacturers to create custom user interfaces (UIs) and add built-in features to some devices. This also puts all developers on an even playing field. Everyone can access the raw Android source code.

The roots of Android

Most people don't know this, but Google didn't start the Android project. The initial Android operating system was created by a small start-up company in Silicon Valley known as Android, Inc., which was purchased by Google in July 2005. The founders of Android, Inc., came from various Internet technology companies such as Danger, Wildfire Communications, T-Mobile, and WebTV. Google brought them into the Google team to help create what is now the full-fledged Android mobile operating system.

Cross-compatibility

Android can run on many devices with different screen sizes and resolutions. Besides being cross-compatible, Android comes with the tools that help you develop cross-compatible applications. Google allows your apps to run only on compatible devices. If your app requires a front-facing camera, for example, only phones with a front-facing camera will be able to see your app in the Android Market. This arrangement is known as *feature detection.* (For more information on publishing your apps to the Android Market, see Chapter 8.)

For Android devices to be certified compatible (devices have to be compatible to have access to the Android Market), they must follow certain hardware guidelines. These guidelines include but are not limited to the following:

- Camera
- Compass
- GPS (Global Positioning System) feature
- Bluetooth transceiver

See the Compatibility Program Overview page at `http://source.android.com/compatibility/overview.html` for specific device configurations that are considered to be compatible. Compatibility ensures that your apps can run on all devices.

Mashup capability

A *mashup* combines two or more services to create an application. You can create a mashup by using the camera and Android's location services, for example, to take a picture with the exact location displayed on the image! It's easy to make a ton of apps by combining services or libraries in new and exciting ways.

With all the APIs that Android includes, it's easy to use two or more of these features to make your own app. You can use a maps API with the contact list to show all your contacts on a map, for example (see "Google APIs," later in this chapter).

Here are a few other mashups to get your brain juices pumping. All this stuff is included for you to use, and it's completely legal and free!

- **Geolocation and social networking:** Social networking is the "in" thing right now. Suppose you want to write an app that tweets your current location every 10 minutes throughout the day. You can, and it's easy. Use Android's location services and a third-party Twitter API (such as iTwitter), and you can do just that.
- **Geolocation and gaming:** Location-based gaming is gaining popularity. It's a great way to really put your users into the game. A game might run a background service to check your current location and compare it with the locations of other users of the game in the same area. If another user is within 1 mile of you, for example, you could be notified, and you could challenge her to a battle. None of this would be possible without a strong platform such as Android and GPS technology.
- **Contacts and Internet:** With all these cool APIs at your disposal, it's easy to make full-featured apps by combining the functionality of two or more APIs. You can combine contacts and the Internet to create a greeting-card app, for example. Or you may just want to add an easy way for your users to contact you from an app or enable users to send the app to their friends. This is all possible with the built-in APIs.

The sky is the limit. All this cool functionality is literally in the palm of your hand. If you want to develop an app that records the geographic location of the device, you can with ease. Android really opens the possibilities by allowing you to tap into these features easily. It's up to you, as the developer, to put them together in a way that can benefit your users.

Developers can do just about anything they want with Android, so be careful. Use your best judgment when creating and publishing apps for mass consumption. Just because you want a live wallpaper that shows you doing the hula in your birthday suit doesn't mean that anyone else wants to see it.

Also, keep privacy laws in mind before you harvest your users' contact info for your own marketing scheme.

Android Programming Basics

You don't have to be a member of Mensa to program Android applications. I'm glad, because otherwise, I wouldn't be writing them! Programming for Android is simple because the default programming language of Android is Java. Although writing Android applications is fairly easy, programming in itself can be a difficult task to conquer.

If you've never programmed before, this book may not be the best place to start. I advise that you pick up a copy of *Beginning Programming with Java For Dummies,* by Barry Burd (Wiley Publishing), to learn the ropes. After you have a basic understanding of Java under your belt, you should be ready to tackle this book.

Although the majority of Android is Java, small parts of the framework aren't. Android also encompasses the XML language as well as basic Apache Ant scripting for build processes. I advise you to have a basic understanding of XML before delving into this book.

If you need an introduction to XML, check out *XML For Dummies,* by Lucinda Dykes and Ed Tittel (Wiley).

If you already know Java and XML, congratulations — you're ahead of the curve!

Java: Your Android programming language

Android applications are written in Java — not the full-blown Java that J2EE developers are used to, but a subset of Java that is sometimes known as the *Dalvik virtual machine.* This smaller subset of Java excludes classes that don't make sense for mobile devices. If you have any experience in Java, you should be right at home.

It may be a good idea to keep a Java reference book on hand, but in any case, you can always Google what you don't understand. Because Java is nothing new, you can find plenty of examples on the Web that demonstrate how to do just about anything.

In Java source code, not all libraries are included. Verify that the package is available to you. If it's not, an alternative is probably bundled with Android that can work for your needs.

Activities

Android applications are made up of one or more activities. Your app must contain at least one activity, but an Android application can contain several. Think of an activity as being a container for your UI, holding your UI as well as the code that runs it. It's kind of like a form, for you Windows programmers out there. I discuss activities in more detail in Chapters 3 and 5.

Intents

Intents make up the core message system that runs Android. An intent is composed of an action that it needs to perform (View, Edit, Dial, and so on) and data. The action is the general action to be performed when the intent is received, and the data is the data to operate on. The data might be a contact item, for example.

Intents are used to start activities and to communicate among various parts of the Android system. Your application can either broadcast an intent or receive an intent.

Sending messages with intents

When you broadcast an intent, you're sending a message telling Android to make something happen. This intent could tell Android to start a new activity from within your application, or it could start a different application.

Registering intent receivers

Just because you send a message doesn't mean that something will happen automatically. You have to register an *intent receiver* that listens for the intent and then tells Android what to do, whether the task is starting a new activity or starting a different app. If many receivers can accept a given intent, a chooser can be created to allow the user to pick the app she wants to use. A classic example is long-pressing an image in an image gallery. *Long-pressing* means clicking something for a long time to bring up a context menu.

By default, various registered receivers handle the image-sharing intents. One of many is e-mail, and another is the messaging application (among various other installed applications). Because you find more than one possible intent receiver, the user is presented with a chooser asking him what he should do: use e-mail, messaging, or another application, as shown in Figure 1-1.

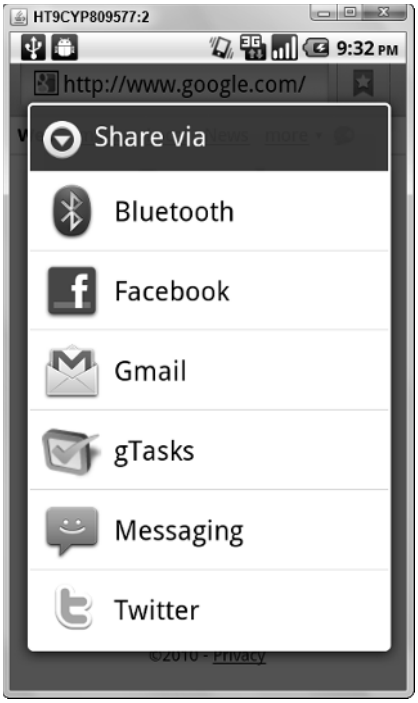

Figure 1-1: A chooser.

If the Android system cannot find a match for the intent that was sent, and a chooser was not created manually, the application will crash due to a *run-time exception:* an unhandled error in the application. Android expects developers to know what they're doing. If you send an intent that a user's Android device doesn't know how to handle, the device crashes. It's best practice to create choosers for intents that don't target other activities within your application.

Cursorless controls

Unlike PCs, which let you use a mouse to move the cursor across the screen, Android devices let you use your fingers to do just about anything a mouse can do. But how do you right-click? Instead of supporting right-clicking, Android has implemented the long press. Press and hold a button, icon, or screen for an extended period of time, and a context menu appears. As a developer, you can create and manipulate context menus. You can allow users to use two fingers on an Android device instead of just one mouse cursor, for example. Keep in mind that fingers come in all sizes, however, and design your user interface accordingly. Make the buttons large enough, with enough spacing, so that even users with large fingers can interact with your apps easily.

Views and widgets

What the heck is a view? A *view* is a basic UI element — a rectangular area on the screen that is responsible for drawing and event handling. I like to think of views as being basic controls, such as a label control in HTML. Here are a few examples of views:

- `ContextMenu`
- `Menu`
- `View`
- `Surface view`

Widgets are more-advanced UI elements, such as check boxes. Think of them as being the controls that your users interact with. Here are a few widgets:

- `Button`
- `CheckBox`
- `DatePicker`
- `DigitalClock`
- `Gallery`
- `FrameLayout`
- `ImageView`
- `RelativeLayout`
- `PopupWindow`

Many more widgets are ready for you to use. Check out the `android.widget` package in the Android documentation at `http://developer.android.com/reference/android/widget/package-summary.html` for complete details.

Asynchronous calls

Who called? I don't know anybody named Asynchronous, do you?

The `AsyncTask` class in Android allows you to run multiple operations at the same time without having to manage a separate thread yourself. `AsyncTask` not only lets you start a new process without having to clean up after yourself, but also returns the result to the activity that started it. This allows you to have a clean programming model for asynchronous processing.

A *thread* is a process that runs separately from and simultaneously with everything else that's happening.

When would you use asynchronous processing? I'm glad you asked! You'd use asynchronous processing for tasks that take a long time — network communication (Internet), media processing, or anything else that might make the user wait. If the user has to wait, you should use an asynchronous call and some type of UI element to let him know that something is happening.

Failing to use an asynchronous programming model can cause users of your application to believe that your application is buggy. Downloading the latest Twitter messages via the Internet takes time, for example. If the network gets slow, and you're not using an asynchronous model, the application will lock up, and the user will assume that something is wrong because the application isn't responding to her interactions. If the application doesn't respond within a reasonable time that the Android OS defines, Android presents an "application not responding" (ANR) dialog box, as shown in Figure 1-2. At that time, the user can decide to wait or to close the application.

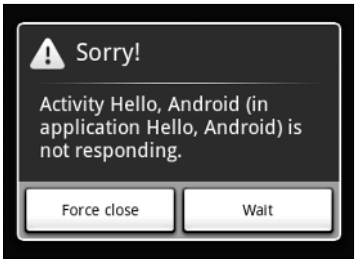

Figure 1-2: An ANR dialog box.

It's best practice to run CPU-expensive or long-running code inside another thread, as described in the Designing for Responsiveness page on the Android developer site (`http://developer.android.com/guide/practices/design/responsiveness.html`).

Background services

If you're a Windows user, you may already know what a *service* is: an application that runs in the background and doesn't necessarily have a UI. A classic example is an antivirus application that usually runs in the background as a service. Even though you don't see it, you know that it's running.

Most music players that can be downloaded from the Android Market run as background services. This is how you can listen to music while checking your e-mail or performing another task that requires the use of the screen.

Hardware Tools

Google exposes a plethora of functionality in Android, thus giving developers (even the independent guys) the tools needed to create top-notch, full-featured mobile apps. Google has gone above and beyond by making it simple to tap into and make use of all the devices' available hardware.

To create a spectacular Android app, you should take advantage of all that the hardware has to offer. Don't get me wrong: If you have an idea for an app that doesn't need hardware assistance, that's okay too.

Android phones come with several hardware features that you can use to build your apps, as shown in Table 1-1.

Table 1-1 Android Device Hardware

Functionality Required	*Hardware*
Where am I?	GPS radio
Which way am I walking?	Built-in compass
Is my phone facing up or down?	Proximity sensor
Is my phone moving?	Accelerometer
Can I use my Bluetooth headphones?	Bluetooth radio
How do I record video?	Camera

Most Android phones are released with the hardware that I discuss in the following sections, but not all devices are created equal. Android is free for hardware manufacturers to distribute, so it's used in a wide range of devices, including some made by small manufacturers overseas (and it's not uncommon for some of these phones to be missing a feature or two).

Also, as the technology advances, phone manufacturers are starting to add features that aren't yet natively supported by Android. But don't worry; manufacturers that add hardware usually offer a software development kit (SDK) that lets developers tap into the device's unique feature. At this writing, HTC's Evo 4G, available from Sprint, is the only Android phone that comes

with a front-facing camera. Because this device is the first of its kind, Sprint has released an SDK that developers can use to access this cool new feature, as well as sample code that lets them implement the feature easily.

Android devices come in all shapes and sizes: phones, tablet computers, and e-book readers. You will find many other implementations of Android in the future, such as Google TV — an Android-powered home appliance — as well as cars with built-in Android-powered touchscreen computers. The engineers behind Android provide tools that let you easily deploy apps for multiple screen sizes and resolutions. Don't worry — the Android team has done all the hard work for you. I cover the basics of screen sizes and densities in Chapter 4.

Touchscreen

Android phones have touchscreens, a fact that opens a ton of possibilities and can enhance users' interaction with your apps. Users can swipe, flip, drag, and pinch to zoom, for example, by moving a finger or fingers on the touchscreen. You can even use custom gestures for your app, which opens even more possibilities.

Android also supports *multitouch,* which means that the entire screen is touchable by more than one finger at a time.

Hardware buttons are old news. You can place buttons of any shape anywhere on the screen to create the UI that's best suited for your app.

GPS

The Android OS combined with a phone's GPS radio allows developers to access a user's location at any given moment. You can track a user's movement as she changes locations. The Foursquare social-networking app is a good example; it uses GPS to determine the phone's location and then accesses the Web to determine which establishment or public place the user is in or near.

Another great example is the Maps application's ability to pinpoint your location on a map and provide directions to your destination. Android combined with GPS hardware gives you access to the phone's exact GPS location. Many apps use this functionality to show you where the nearest gas station, coffeehouse, or even restroom is located. You can even use the maps API to pinpoint the user's current location on a map.

Accelerometer

Android comes packed with accelerometer support. An *accelerometer* is a device that measures acceleration. That sounds cool and all, but what can you do with it? If you want to know whether the phone is moving or being shaken, or even the direction in which it's being turned, the accelerometer can tell you.

You may be thinking, "Okay, but why do I care whether the phone is being shaken or turned?" Simple! You can use that input as a way to control your application. You can do simple things like determine whether the phone has been turned upside down and do something when it happens. Maybe you're making a dice game and want to immerse your users in the game play by having them shake the phone to roll the dice. This is the kind of functionality that is setting mobile devices apart from typical desktop personal computers.

SD Card

Android gives you the tools you need to access (save and load) files on the device's *SD Card* — a portable storage medium that you can insert into various phones and computers. If a device is equipped with an SD Card, you can use it to store and access files needed by your application. Android 2.2 allows you to install apps on the SD Card, but maybe your users have phones that don't get Android 2.2. Just because some users don't have the option of installing apps on the SD Card doesn't mean that you have to bloat your app with 20MB of resources and hog the phone's limited built-in memory. You can download some or all of your application's resources from your Web host and save them to the phone's SD Card. This makes your users happy and less likely to uninstall your app when space is needed.

Not all devices come with an SD Card installed, although most do. Always make sure that the user has an SD Card installed and that adequate space is available before trying to save files to it.

Software Tools

Various Android tools are at your disposal while writing Android applications. In the following sections, I outline some of the most popular tools that you will use in your day-to-day Android development process.

Internet

Thanks to the Internet capabilities of Android devices, real-time information is easy to obtain. As a user, you can use the Internet to see what time the next movie starts or when the next commuter train arrives. As a developer, you can use the Internet in your apps to access real-time, up-to-date data such as weather, news, and sports scores. You can also use the Web to store some of your application's assets, which is what Pandora and YouTube do.

Don't stop there. Why not offload some of your application's intense processes to a Web server when appropriate? This can save a lot of processing time in some cases and also helps keep your Android app streamlined. This arrangement is called *client–server computing* — a well-established software architecture in which the client makes a request to a server that is ready and willing to do something. The built-in Maps app is an example of a client accessing map and GPS data from a Web server.

Audio and video support

The Android OS makes including audio and video in your apps a breeze. Many standard audio and video formats are supported. Including multimedia content in your apps couldn't be any easier. Sound effects, instructional videos, background music, streaming video, and audio from the Internet can all be added to your app with little to no pain. Be as creative as you want to be. The sky is the limit.

Contacts

Your app can access user contacts that are stored on the phone. You can use this feature to display the contacts in a new or different way. Maybe you don't like the built-in Contacts application. With the ability to access the contacts stored on the phone, nothing is stopping you from writing your own. Maybe you write an app that couples the contacts with the GPS system and alerts the user when she is close to one of the contacts' addresses.

Use your imagination, but be responsible. You don't want to use contacts in a malicious way (see the next section).

Security

Android allows your apps to do a lot! Imagine if someone released an app that went through the contact list and sent the entire list to a server somewhere for malicious purposes. This is why most of the functions that modify the user's device or access its protected content need to have permissions to work. Suppose that you want to download an image from the Web and save it to the SD Card. To do so, you need to get permission to use the Internet so that you can download the file. You also need permission to save files to the SD Card. Upon installation of the application, the user is notified of the permissions that your app is requesting. At that point, the user can decide whether he wants to proceed with the installation. Asking for permission is as easy as implementing one line of code in your application's manifest file, which I cover in Chapter 3.

Google APIs

The Android OS isn't limited to making phone calls, organizing contacts, or installing apps. You have much more power at your fingertips. As a developer, you can integrate maps into your application. To do so, you have to use the maps APIs that contain the map widgets.

Pinpointing locations on a map

Perhaps you want to write an app that displays your current location to your friends. You could spend hundreds of hours developing a mapping system — or you could just use the Android Maps API. Google provides the Android Maps API, which you can use in your app, and just like everything else in Android, it's free! You can embed and use the API in your application to show your friends where you are; it won't take hundreds of hours or cost you a cent. Imagine all the juicy map goodness with none of the work developing it. Using the maps API, you can find just about anything with an address; the possibilities are endless. Display your friend's location, the nearest grocery store, or the nearest gas station — anything or anyplace with an address.

Getting around town with navigation

Showing your current location to your friends is cool, but wait — there's more! The Android Maps API can also access the Google Navigation API. Now you can pinpoint your location and also show your users how to get to that location.

The KISS principle

It's easy to overthink and overcomplicate things when developing applications. The hardest part is to remember the KISS (Keep It Simple, Stupid) principle. One way to overly complicate your code is to just dive in without understanding all the built-in APIs and knowing what they do. You can go that route, but doing so may take more time than just glossing over the Android documentation. You don't have to memorize it, but do yourself a favor and take a look at the documentation. You'll be glad you did when you see how easy it is to use the built-in functionality and how much time it can save you. You can easily write multiple lines of code to do something that takes only one line. Changing the volume of the media player or creating a menu is a simple process, but if you don't know the APIs, you may end up rewriting them and in the end causing yourself problems.

When I started with my first app, I just dived in and wrote a bunch of code that managed the media player's volume. If I'd just looked into the Android documentation a little more, I'd have known that I could handle this with one line of code that's strategically placed inside my application. The same thing goes for the menu. I wrote a lot of code to create a menu, and if I'd only known that a menu framework already existed, it would have saved me several hours.

Another way to really muck things up is to add functionality that isn't needed. Most users want the easiest way to do things, so don't go making some fancy custom tab layout when a couple of menu items will suffice. Android comes with enough built-in controls (widgets) that you can use to accomplish just about anything. Using the built-in controls makes your app that much easier for your users to figure out because they already know and love these controls.

Messaging in the clouds

You may be thinking — clouds, I don't see a cloud in the sky! Well, I'm not talking about those kinds of clouds. The Android Cloud to Device Messaging framework allows you to send a notification from your Web server to your app. Suppose that you store your application's data in the cloud and download all the assets the first time your app runs. But what if you realize after the fact that one of the images is incorrect? For the app to update the image, it needs to know that the image changed. You can send a cloud-to-device message (a message from the Web to the device) to your app, letting it know that it needs to update the image. This works even if your app is not running. When the device receives the message, it dispatches a message to start your app so that it can take the appropriate action.